WEEK 2

Q – N meetings in one room

```
class Solution
{
    public:

    int maxMeetings(int start[], int end[], int n)
    {
        pair<int, int> a[n + 1];
    int i;
    for (i = 0; i < n; i++) {
        a[i].first = end[i];
        a[i].second = i;
    }
   sort(a, a + n);
  int time_limit = a[0].first;
    vector<int> m;
    m.push_back(a[0].second + 1);

    for (i = 1; i < n; i++) {
        if (start[a[i].second] > time_limit) {
            m.push_back(a[i].second + 1);
             time_limit = a[i].first;
        }
    }
      return m.size();

}
```

Q – Reverse words in string

```
class Solution {
public:
    string reverseWords(string s) {
        stack<string>st;
        string str = "";
        for(int i = 0;i<s.length();i++){
            if(s[i] != ' ' ){
                str += s[i];
                continue;
            }
            else{
                if(str!="")
                  st.push(str);
                str="";
            }
        }
        if(str!=""){
            st.push(str);
        }
        string ans ="";
        while(!st.empty()){
            ans += st.top() + " ";
            st.pop();
        }
        ans.pop_back();
        return ans;
    }
    };
```

Q – Longest Palindrome in a String

```
 class Solution {
public:
    string longestPalindrome(string s) {
        int start=0;
        int end=0;
        int n=s.size();
        for(int i=1;i<n;i++)
        {
            int l=i-1;
            int h=i;
            while(l>=0 && h<n && s[l]==s[h])
            {
                if(end-start < h-l )
                {
                    start=l;
                    end=h;
                }
                l--;
                h++;
            }

            l=i-1;
            h=i+1;
            while(l>=0 && h<n && s[l]==s[h])
            {
                if(end-start < h-l )
                {
```

```
                start=l;
                end=h;
            }
            l--;
            h++;
          }
       }
       string ans="";
       for(int i=start;i<=end;i++)
          ans+=s[i];

       return ans;
    }
};
```

Q- Roman To Integers

```
class Solution {
public:
    int romanToInt(string s) {
       unordered_map<char, int> rTi = {{'I', 1},
                                       {'V', 5},
                                       {'X', 10},
                                       {'L', 50},
                                       {'C', 100},
                                       {'D', 500},
                                       {'M', 1000}};
       int i, ln = s.size(), ans = rTi[s.back()];
       if(ln==0 || ln>15)
          return 0;
```

```
        else {
            for(i=ln-2; i>-1; i--) {
                if(rTi[s[i]] < rTi[s[i+1]])
                    ans -= rTi[s[i]];
                else
                    ans += rTi[s[i]];
            }
        }
        return ans;
    }
};
```

Q- String into Integer

```
class Solution {
public:
    int myAtoi(string s) {
    double ans = 0;
    bool isNegative = false;
    int index = 0;
    int n = s.size();
    if(s == "" || n == 0){
        return 0;
    }
   while(index < n && s[index] == ' '){
        index++;
    }
    if(s[index] == '+' || s[index] == '-'){
        if(s[index] == '-'){
            isNegative = true;
```

```
        }
        index++;
    }

    for(int i = index;i < n;i++){
      if(s[i] < '0' || s[i] > '9'){
          break;
        }
        int digit = s[i] - '0';
        ans = ans * 10 + digit;
    }
    if(isNegative){
        ans = -ans;
    }
    if(ans < INT_MIN){
        ans = INT_MIN;
    }
    if(ans > INT_MAX){
        return INT_MAX;
    }
    return (int)ans;
    }
};
```

Q- Longest Common Prefix

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        string a = "";
```

```
    if(strs[0] == a){
        return a;
    }

            if(strs[0][0] != strs[strs.size()-1][0]){
        return a;
    }
    a.push_back(strs[0][0]);

    string b = strs[0];
    string c = strs[strs.size()-1];

    for(int i = 1 ; i < b.size() ; i++){
        if(b[i] == c[i]){
            a.push_back(b[i]);
        }
        else{return a;}
    }
    return a;
    }
};
```

Q- LCA

```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if (root == NULL || root == p || root == q) {
            return root;
```

```
        }
        TreeNode* left = lowestCommonAncestor(root->left, p, q);
        TreeNode* right = lowestCommonAncestor(root->right, p, q);

        if(left == NULL) {
            return right;
        }
        else if(right == NULL) {
            return left;
        }
        else {
            return root;
        }
    }
};
```

Q-two trees are identical or not

```
bool isIdentical(node * node1, node * node2) {
  if (node1 == NULL && node2 == NULL)
    return true;
  else if (node1 == NULL || node2 == NULL)
    return false;

  return ((node1 -> data == node2 -> data) && isIdentical(node1 -> left, node2 -> left) &&
isIdentical(node1 -> right, node2 -> right));
}
```

Q- Zig Zag Traversal of Binary Tree

```
vector < vector < int >> zigzagLevelOrder(Node * root) {
```

```
vector < vector < int >> result;
if (root == NULL) {
  return result;
}

queue < Node * > nodesQueue;
nodesQueue.push(root);
bool leftToRight = true;

while (!nodesQueue.empty()) {
  int size = nodesQueue.size();
  vector < int > row(size);
  for (int i = 0; i < size; i++) {
    Node * node = nodesQueue.front();
    nodesQueue.pop();
    int index = (leftToRight) ? i : (size - 1 - i);

    row[index] = node -> val;
    if (node -> left) {
      nodesQueue.push(node -> left);
    }
    if (node -> right) {
      nodesQueue.push(node -> right);
    }
  }
  leftToRight = !leftToRight;
  result.push_back(row);
}
return result;
```

```
}
```

Q- Boundary Traversal of Binary Tree

```
bool isLeaf(node * root) {
  return !root -> left && !root -> right;
}

void addLeftBoundary(node * root, vector < int > & res) {
  node * cur = root -> left;
  while (cur) {
    if (!isLeaf(cur)) res.push_back(cur -> data);
    if (cur -> left) cur = cur -> left;
    else cur = cur -> right;
  }
}
void addRightBoundary(node * root, vector < int > & res) {
  node * cur = root -> right;
  vector < int > tmp;
  while (cur) {
    if (!isLeaf(cur)) tmp.push_back(cur -> data);
    if (cur -> right) cur = cur -> right;
    else cur = cur -> left;
  }
  for (int i = tmp.size() - 1; i >= 0; --i) {
    res.push_back(tmp[i]);
  }
}

void addLeaves(node * root, vector < int > & res) {
```

```
  if (isLeaf(root)) {
    res.push_back(root -> data);
    return;
  }
  if (root -> left) addLeaves(root -> left, res);
  if (root -> right) addLeaves(root -> right, res);
}

vector < int > printBoundary(node * root) {
  vector < int > res;
  if (!root) return res;

  if (!isLeaf(root)) res.push_back(root -> data);

  addLeftBoundary(root, res);
  addLeaves(root, res);

  addRightBoundary(root, res);
  return res;
}
```

Q- Maximum Sum Path in Binary Tree

```
int findMaxPathSum(node * root, int & maxi) {
  if (root == NULL) return 0;

  int leftMaxPath = max(0, findMaxPathSum(root -> left, maxi));
  int rightMaxPath = max(0, findMaxPathSum(root -> right, maxi));
  int val = root -> data;
  maxi = max(maxi, (leftMaxPath + rightMaxPath) + val);
```

```
  return max(leftMaxPath, rightMaxPath) + val;

}

int maxPathSum(node * root) {
  int maxi = INT_MIN;
  findMaxPathSum(root, maxi);
  return maxi;

}
```

Q- Construct A Binary Tree from Inorder and Preorder Traversal

```
node * constructTree(vector < int > & preorder, int preStart, int preEnd, vector
 < int > & inorder, int inStart, int inEnd, map < int, int > & mp) {
  if (preStart > preEnd || inStart > inEnd) return NULL;

  node * root = newNode(preorder[preStart]);
  int elem = mp[root -> data];
  int nElem = elem - inStart;

  root -> left = constructTree(preorder, preStart + 1, preStart + nElem, inorder,
  inStart, elem - 1, mp);
  root -> right = constructTree(preorder, preStart + nElem + 1, preEnd, inorder,
  elem + 1, inEnd, mp);

  return root;
}
```

```
node * buildTree(vector < int > & preorder, vector < int > & inorder) {
  int preStart = 0, preEnd = preorder.size() - 1;
  int inStart = 0, inEnd = inorder.size() - 1;

  map < int, int > mp;
  for (int i = inStart; i <= inEnd; i++) {
    mp[inorder[i]] = i;
  }

  return constructTree(preorder, preStart, preEnd, inorder, inStart, inEnd, mp);
}
```

Q- Construct Binary Tree from Inorder and Postorder

```
class Solution {
public:
   TreeNode* dfs(vector<int>v, vector <int>& p){
    if(!p.size()){
        return NULL;
    }
    vector<int> l;
    vector<int> r;
int t=0;

    int k= p[p.size()-1];
   for(int i=0;i<v.size();i++){
        if(v[i]==k)
            t=1;
        else if(t==0)
```

```
            l.push_back(v[i]);


        else
            r.push_back(v[i]);
    }
    if(t==1){
        TreeNode* node = new TreeNode(k);
    p.pop_back();


        node->right=dfs(r,p);
        node->left=dfs(l,p);


        return node;
    }
    else{
        return NULL;
    }
}
TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {

return dfs(inorder,postorder);


}
};
```